МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедры МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Линейные структуры данных: стек, очередь и дек

Студентка гр. 8304 _____ Мельникова О.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы

Ознакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование очереди для решения практических задач.

Задание

Вариант 3-в

Рассматриваются следующие типы данных:
type *имя* = (*Анна, …, Яков*);

дети = array[*имя, имя*] of Boolean;

file of имя.

Задан массив Д типа дети (Д[x, y] = true, если человек по имени y является ребенком человека по имени x). Для введенного пользователем имени И записать в файл П типа потомки имена всех потомков человека с именем И в следующем порядке: сначала имена всех его детей, затем всех его внуков, затем всех правнуков и т.д.

Класс Queue

Специально для очереди определена следующая глобальная переменная:

```
#define START_SIZE 10
```

Очередь была реализована на основе класса Queue, объекты которого описаны в табл. 2. Методы данного класса описаны в табл. 3.

Таблица 2 — Описание объектов класса Queue

Объект	Тип	Описание
int * data	Private	Указатель под массив имен.
int size	Private	Текущий размер очереди.

Таблица 3 — Методы класса Queue

Метод	Тип	Выходные данные	Входные данные	Описание
Queue ()	Public	-	-	Конструктор очереди. Выделяет память под *data* и устанавливает *size* в 0.
~Queue()	Public	-	-	Деструктор очереди.
Push	Public	-	int val — значение для записи;	Добавление элемента в конец очереди
Pop	Public	int	-	Получение первого элемента из очереди, удаление этого элемента из очереди со смещением очереди в сторону удаляемого элемента.

Окончание таблицы 3

get_elem	Public	int	int index- индекс	Возвращает элемент по индексу в очереди
isEmpty	Public	int	-	Возвращает 1 в случае пустой очереди и 0 в обратном случае
size	Public	int	-	Возвращает размер очереди

Считывание и функция GetManID

Считывание выполняет функция `ReadAndWritePeople` и `ConsoleReadAndWritePeople`, которые принимают на вход вектор имен и два вектора индексов, показывающих родство, а также количество элементов в этих векторах и очередь `queue`. `ReadAndWritePeople` кроме того принимает аргумент командной строки – название файла.

В цикле считываются строки с именами, где первое имя – это родитель, а второе – ребенок. При считывании имен происходит поиск в векторе с именами, и если текущего имени там нет, то записываем его. Кроме того заполняем векторы индексов (первое имя – каждое последующее). Для быстрого поиска индекса в векторе имен создана функция `GetManID`.

Считывание пар родственников происходит до того как программа встретит знак “-”. Затем считывается имя для поиска и записывается в очередь.

Функция поиска поколений

Функция func1 принимает вектор с именами, вектора с индексами, показывающими родство, количество элементов, очередь, файл для записи и глубину.

В теле функции создается вторая очередь, в которую записываются все дети имен, индексы которых указаны в текущей очереди. В зависимости от глубины печатаются имена по индексам из созданной очереди, с указанием поколения.

Далее происходит рекурсивный вызов функции, помимо неизменяемых элементов в нее подается вторая очередь и глубина, увеличенная на 1.

Функция завершается, когда создаваемая очередь оказывается пустой после записи детей для предыдущей.

Тестирование

Содержимое файла:	**Содержимое файла result.txt**
ivan peter john anna peter peter jack jack gorge gorge olga - ivan	Дети: peter Внуки: jack Правнуки: gorge Праправнуки: olga
Содержимое файла:	**Содержимое файла result.txt**
Lena Igor Igor Dasha Lena Olya Igor Andrey - Lena	Дети: Igor Olya Внуки: Dasha Andrey
Содержимое файла:	**Содержимое файла result.txt**
-	

Olya	
Содержимое файла: Kate Vadim - Vadim	**Содержимое файла result.txt**
Содержимое файла: Roma9 M7 Alex Igor Alex Roma9 - Alex	**Содержимое файла result.txt** Дети: Igor

Вывод

В данной работе было создана программа, которая по данным родственным связям находит всех детей, внуков, правнуков и т.д.

Исходный код программы

Queue.cpp

```
#include <iostream>
#define START_SIZE 10

class Queue{
    public:
        Queue()  //Конструктор выделяет память
под данные.
        {
            _size = 0;
```

```
            data = new int[START_SIZE];
        }
        ~Queue() {}
        //Деструктор пуст,так как подается в
функции.

        void push(int val)    //Пуш элемента
        {
            resize();
            //Увеличиваем память, если нужно.
            data[_size] = val;
            _size++;
            //Запишем новый элемент
        }

        int pop()    //Поп первогоэлемента со
смещением всех элементов влево
        {
            int poped = data[0];
            for(int i = 0; i < _size - 1; i++)
data[i] = data[i+1];
            _size--;
            return poped;
        }

        int get_elem(int index)   //Поп
первогоэлемента со смещением всех элементов
влево
        {
            return data[index];
        }

        int isEmpty()    //Проверяет очередь на
пустоту
```

```
        {
            return !_size;
        }

        int size()
        {
            return _size;
        }

    private:
        int *data;
        //Указатель под массив имен
        int _size;
        //Размер очереди

        void resize()    //Динамическое
увеличение массива data
        {
            if(_size % START_SIZE == 0 &&
_size)
            {
                int *pTmp = new int[_size +
10];
                for(int i = 0; i < _size; i++)
                pTmp[i] = data[i];
                delete[] data;
```

```
#include <fstream>
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <vector>
#include <regex.h>
#include "Queue.cpp"

#define SIZE 1000

unsigned char IsStringOK(char* str){
    unsigned char i = 0;
    for(i = 0; i<strlen(str); i++){
        if(!isalpha(str[i])) return 0;
    }
    return 1;
}

int GetManID(std::string name,
std::vector<std::string>* people, int
peopleCount)
{
    for (int i = 0; i < peopleCount; i++)
        if ((*people)[i] == name)
            return i;
    //в цикле по people сравниваем name с
people[i]. если равно, то возвращаем i
    return -1;
}

void
ConsoleReadAndWritePeople(std::vector<std::strin
g>* people, std::vector<int>* parents,
```

```
                people->push_back(name);
                //people[*peopleCount] = name;
                (*peopleCount)++;
            }

            if(flagChild){      // если считали
второе и последующие имена в строке, то
добавляем отношение
                parents->push_back(parentID);
//(индекс первого имени в строке – индекс
считанного имени)
                children-
>push_back(GetManID(name, people,
*peopleCount));
                (*relationsCount)++;
            }
            else
            {
                parentName = name;
                parentID = GetManID(parentName,
people, *peopleCount);  //функция поиска индекса
имени в массиве имен
            }

            name = new char[20];
            name = strtok (NULL, " ");
            flagChild = 1;
        }
    }
    std::cout<<"Введите имя, для которого
совершать поиск";
    std::string name1;
    std::cin>>name1;  // если не удалось –
выход
```

```
    queue->push(GetManID(name1, people,
*peopleCount));
    if(GetManID(name1, people, *peopleCount)==-
1) exit;

}

void
ReadAndWritePeople(std::vector<std::string>*
people, std::vector<int>* parents,
std::vector<int>* children, int* peopleCount,
int* relationsCount, Queue* queue, char* argv1)
{
    // в цикле считываем строки
    std::ifstream fin;
    std::string str;
    regex_t regex;
    int reti1;
    reti1 = regcomp(&regex, ".\\.txt", 0);
    reti1 = regexec(&regex, argv1, 0, NULL, 0);
    if (!reti1){
        fin.open(argv1);
    }else{
            std::cout<<"Некорректные названия
файла (расширение txt)"<<std::endl;
            exit(0);
    }
    while(str!="-")
    {
        getline(fin, str);
        if(str==""){
             exit(1);
        }
        if(str=="-") continue;
```

```
        char* cstr = new char[str.length()+1];
        strcpy(cstr, str.c_str());
        char* name = new char[20];
        name = strtok (cstr, " ");

        int flagChild = 0;
        std::string parentName;
        int parentID;

        // ищем имена в people[]
        // если имени нет - добавить

        while ((name != NULL) &&
(IsStringOK(name)) && (flagChild<2))
        {
            int flag = 1;
            for(int i = 0; i < *peopleCount; i+
+){
                if(strcmp(name, (*people)
[i].c_str()) == 0) { flag = 0;    break; }
            }
            if(flag){
                people->push_back(name);
                //people[*peopleCount] = name;
                (*peopleCount)++;
            }

            if(flagChild){      // если считали
второе и последующие имена в строке, то
добавляем отношение
                parents->push_back(parentID);
//(индекс первого имени в строке - индекс
считанного имени)
```

```
                children-
>push_back(GetManID(name, people,
*peopleCount));
                (*relationsCount)++;

            }
            else
            {
                parentName = name;
                parentID = GetManID(parentName,
people, *peopleCount);  //функция поиска индекса
имени в массиве имен
            }

            name = new char[20];
            name = strtok (NULL, " ");
            flagChild++;
        }
    }
    std::string name1;
    getline(fin, name1);  // если не удалось -
выход
    queue->push(GetManID(name1, people,
*peopleCount));
    if(GetManID(name1, people, *peopleCount)==-
1) exit;

}


int func1(std::vector<std::string> people,
std::vector<int> parents, std::vector<int>
children, int relationsCount, std::ofstream&
fout, Queue* prev_queue, int k){
```

```
    Queue *queue = new Queue;
    for(int i = 0; i<(prev_queue->size()); i++)
{
        for(int j = 0; j<relationsCount; j++){
            if((prev_queue-
>get_elem(i))==parents[j]){
                queue->push(children[j]);
            }
        }
    }
    //если  massive пуст, то return 0;
    if(queue->isEmpty()){
        fout<<"";
        return 0;
    }

    if(k==0) fout<<"\nДети:"<<std::endl;
    else if(k==1) fout<<"\nВнуки:"<<std::endl;
    else{
        fout<<"\nПра";
        for(int a=0; a<k-2; a++){
            fout<<"пра";
        }
        fout<<"внуки:"<<std::endl;
    }
    for(int s=0; s<(queue->size()); s++){
        fout<<people[queue-
>get_elem(s)]<<std::endl;
    }
    k++;
    func1(people, parents, children,
relationsCount, fout, queue, k);
}
```

```
int main(int argc, char *argv[])
{
    int flag=1;
    std::vector<std::string> people;
    Queue *queue = new Queue;
    std::vector<int> parents;   //индексы имен
из people в двух массивах показывают родство
    std::vector<int> children;
    int peopleCount = 0;  //количество жителей
    int relationsCount = 0;  //количество
связей


    if(argv[1]!=NULL){
        ReadAndWritePeople(&people, &parents,
&children, &peopleCount, &relationsCount, queue,
argv[1]);
    }else{
        std::cout<<"\nВы не ввели или ввели
неправильно аргумент командной строки.\nПервый
аргумент командной строки – файл с расширением
txt, из которого считывается список.\nДля
продолжения введите 1, иначе 0."<<std::endl;

        std::cin>>flag;
        if(flag==0) return 0;

        ConsoleReadAndWritePeople(&people,
&parents, &children, &peopleCount,
&relationsCount, queue);
    }
```

```
    std::ofstream fout("result.txt",
std::ios::out);
    if(parents.empty()){
        fout<<"";
        return 0;
    }
    func1(people, parents, children,
relationsCount, fout, queue, 0);


    return 0;
}
```